The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
邢浩、何嘉豪、黄迦密

Supervisor:
Mingkui Tan

Student ID：201530613221（邢浩）201530361276（何嘉豪）201537611718（黄迦密）

Grade:

Undergraduate

December 9, 2017

Face Classification Based on AdaBoost Algorithm

Abstract—

I. Introduction

A. Motivation of Experiment

1. Understand Adaboost further
2. Get familiar with the basic method of face detection
3. Learn to use Adaboost to solve the face classification problem,and combine the theory with the actual project
4. Experience the complete process of machine learning

B. Dataset

1. This experiment provides 1000 pictures, of which 500 are human face RGB images, stored in *datasets/original/face*; the other 500 is a non-face RGB images, stored in *datasets/original/nonface*.
2. The dataset is included in the example repository. Please download it and divide it into training set and validation set.

C. Environment for Experiment

python3, at least including following python package: sklearn, numpy, matplotlib, pickle, PIL.
It is recommended to install anaconda3 directly, which has built-in python package above.
PyCharm Community Integrated Development Environment (optional)

II. Methods and Theory

AdaBoost is an iterative algorithm. The core idea of AdaBoost is to train different classifiers, ie weak classifiers, against the same training set, and then combine these weak classifiers to construct a stronger final classifier.
The algorithm itself is to change the distribution of data to determine the weight of each sample based on the correct classification of each sample in each training set and the accuracy of the last overall classification. The new data of the modified weights are sent to the lower classifier for training, and then the classifiers obtained by each training are fused together to be the final decision classifier

III. Experiment

A. Experiment Step

1. Read data set data. The images are supposed to converted into a size of 24 * 24 grayscale, the number and the proportion of the positive and negative samples is not limited, the data set label is not limited.
2. Processing data set data to extract NPD features. Extract features using the NPDFeature class in feature.py. (Tip: Because the time of the pretreatment is relatively long, it can be pretreated with pickle function library dump () save the data in the cache, then may be used load () function reads the characteristic data from cache.)
3. The data set is divisded into training set and calidation set, this experiment does not divide the test set.
4. Write all *AdaboostClassifier* functions based on the reserved interface in *ensemble.py*. The following is the guide of *fit* function in the *AdaboostClassifier* class:
   4.1 Initialize training set weights w, each training sample is given the same weight.
   4.2Training a base classifier , which can be sklearn.tree library DecisionTreeClassifier(note that the training time you need to pass the weight w as a parameter).
   4.3 Calculate the classification error rate w of the base classifier on the training set.
   4.4 Calculate the parameter w according to the classification error rate w.
   4.5 Update training set weights w.
   4.6 Repeat steps 4.2-4.6 above for iteration, the number of iterations is based on the number of classifiers.
5. Predict and verify the accuracy on the validation set using the method in AdaboostClassifier and use classification_report () of the sklearn.metrics library function writes predicted result to *report.txt* .
6. Organize the experiment results and complete the lab report (the lab report template will be included in the example repository).

B. Code

feature.py

```
import numpy


class NPDFeature():
    """It is a tool class to extract the NPD features.

    Attributes:
        image: A two-dimension ndarray indicating grayscale
image.
        n_pixels: An integer indicating the number of image total
pixels.
        features: A one-dimension ndarray to store the extracted
NPD features.
    """
    __NPD_table__ = None

    def __init__(self, image):
        '''Initialize NPDFeature class with an image.'''
        if NPDFeature.__NPD_table__ is None:
            NPDFeature.__NPD_table__ =
NPDFeature.__calculate_NPD_table()
        assert isinstance(image, numpy.ndarray)
        self.image = image.ravel()
        self.n_pixels = image.size
        self.features = numpy.empty(shape=self.n_pixels *
(self.n_pixels - 1) // 2, dtype=float)

    def extract(self):
        '''Extract features from given image.

        Returns:
            A one-dimension ndarray to store the extracted NPD
features.
        '''
        count = 0
        for i in range(self.n_pixels - 1):
            for j in range(i + 1, self.n_pixels, 1):
                self.features[count] =
NPDFeature.__NPD_table__[self.image[i]][self.image[j]]
                count += 1
        return self.features

    @staticmethod
    def __calculate_NPD_table():
        '''Calculate all situations table to accelerate feature
extracting.'''
        print("Calculating the NPD table...")
        table = numpy.empty(shape=(1 << 8, 1 << 8), dtype=float)
        for i in range(1 << 8):
            for j in range(1 << 8):
                if i == 0 and j == 0:
                    table[i][j] = 0
                else:
                    table[i][j] = (i - j) / (i + j)
        return table
```

ensemble.py

```
import pickle
from sklearn.metrics import classification_report
import pandas as pd
import numpy as np
class AdaBoostClassifier:
    '''A simple AdaBoost Classifier.'''

    def __init__(self, weak_classifier, n_weakers_limit=10):
        '''Initialize AdaBoostClassifier

        Args:
            weak_classifier: The class of weak classifier, which is
recommend to be sklearn.tree.DecisionTreeClassifier.
            n_weakers_limit: The maximum number of weak
classifier the model can use.
        '''
        self.estimator=weak_classifier
        self.estimators=[]
        self.n_estimators=n_weakers_limit
        self.sample_weight=None
        self.alphas=[]
        self.learning_rate=1
        self.alpha=1
        pass

    def is_good_enough(self):
        '''Optional'''
        pass

    def fit(self,X,y):
        '''Build a boosted classifier from the training set (X, y).

        Returns:
            X: An ndarray indicating the samples to be trained,
which shape should be (n_samples,n_features).
            y: An ndarray indicating the ground-truth labels
correspond to X, which shape should be (n_samples,1).
        '''
        for iboost in range(self.n_estimators):
            if self.sample_weight is None:
                # Initialize weights to 1 / n_samples
                self.sample_weight = np.empty(X.shape[0],
dtype=np.float64)
                self.sample_weight[:] = 1. / X.shape[0]
            else:
                # Normalize existing weights
                self.sample_weight = self.sample_weight /
self.sample_weight.sum(dtype=np.float64)

            self.estimator.fit(X, y,
sample_weight=self.sample_weight.reshape(-1))
            y_predict = self.estimator.predict(X).reshape(-1,1)
            incorrect = y_predict != y
            estimator_error = np.mean(np.average(incorrect,
weights=self.sample_weight, axis=0))
            if estimator_error<=0:
                self.alphas.append(self.alpha)
                self.estimators.append(self.estimator)
                continue
            else:

self.alpha=0.5*np.log(1.*(1-estimator_error)/estimator_error)

self.sample_weight=self.sample_weight.reshape(-1,1)*np.exp(
-self.alpha*y.reshape(-1,1)*y_predict.reshape(-1,1))
                self.alphas.append(self.alpha)
                self.estimators.append(self.estimator)
        return X,y
    def predict_scores(self, X):
        '''Calculate the weighted sum score of the whole base
classifiers for given samples.

        Args:
            X: An ndarray indicating the samples to be predicted,
which shape should be (n_samples,n_features).

        Returns:
            An one-dimension ndarray indicating the scores of
differnt samples, which shape should be (n_samples,1).
        '''

scores=np.empty(X.shape[0],dtype=np.float64).reshape(-1,1)
        for iboost in range(self.n_estimators):
            scores=np.concatenate(
                (1.*self.alphas[iboost]*

self.estimators[iboost].predict_proba(X)[:,1].reshape(-1,1)

/(self.estimators[iboost].predict_proba(X)[:,1].reshape(-1,1)+
                   self.estimators[iboost].predict_proba(X)[:,
0].reshape(-1, 1))
                 ,scores),axis=1)
        return np.average(scores[:,0:-1],axis=1).reshape(-1,1)

    def predict(self, X, threshold=0.5):
        '''Predict the catagories for geven samples.

        Args:
            X: An ndarray indicating the samples to be predicted,
which shape should be (n_samples,n_features).
            threshold: The demarcation number of deviding the
samples into two parts.

        Returns:
            An ndarray consists of predicted labels, which shape
should be (n_samples,1).
        '''
        score=self.predict_scores(X)
        df = pd.DataFrame(score)
        df[1] = df[0].apply(lambda x: 1 if x > threshold else -1)
        return np.array(df[1]).reshape(-1,1)

    @staticmethod
    def save(model, filename):
        with open(filename, "wb") as f:
            pickle.dump(model, f)

    @staticmethod
    def load(filename):
        with open(filename, "rb") as f:
            return pickle.load(f)
```

train.py

```
import feature
import ensemble
from PIL import Image
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
import codecs

size=24,24
path='/home/qian/iNet/PycharmProjects/ML2017-lab-03-mast
er/datasets/original/face/face_'
pathnon='/home/qian/iNet/PycharmProjects/ML2017-lab-03-
master/datasets/original/nonface/nonface_'
face_f=[]
nface_f=[]
num=30
for i in range(num):
    name=path+'%03d'%i+'.jpg'
    obj = Image.open(name).convert('L')
    obj.thumbnail(size, Image.ANTIALIAS)
    npd=feature.NPDFeature(np.array(obj))
    n = npd.extract()
    face_f.append(n.tolist())
    name = pathnon + '%03d' % i + '.jpg'
    obj = Image.open(name).convert('L')
    obj.thumbnail(size, Image.ANTIALIAS)
    npd = feature.NPDFeature(np.array(obj))
    n = npd.extract()
    nface_f.append(n.tolist())
print('Calculat NPD success.')
p=np.ones(num)
n=-p
label=np.concatenate((p.reshape(-1,1),n.reshape(-1,1)),axis=0)
data=np.concatenate((face_f,nface_f),axis=0)
weak=DecisionTreeClassifier()
clf=ensemble.AdaBoostClassifier(weak_classifier=weak,n_we
akers_limit=10)
from sklearn.model_selection import train_test_split
training_X, validation_X,training_Y,validation_Y =
train_test_split(data,label,test_size=0.1)
clf.fit(training_X,training_Y)
y_pred=clf.predict(validation_X)
incorrect=y_pred!=validation_Y
#print("acc on validation:",1-np.mean(np.average(incorrect,
axis=0)))
y_pred=y_pred.reshape(-1).tolist()
y_true=validation_Y.reshape(-1).tolist()
target_names=['face','nonface']
fout = codecs.open('report.txt','w','utf-8')
result=classification_report(y_true, y_pred,
target_names=target_names)
fout.write(result)
print(result)
fout.close()
```

IV. CONCLUSION

Through this experiment, I was further acquainted with the principle of adaboost, learned to deal with image features, a very meaningful experiment